VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

# Multiplatformní aplikace pro podporu výuky síťové analýzy

# Multi-platform Application for Network Analysis Teaching Support

Student: Lukáš Josefus

Vedoucí bakalářské práce: doc. RNDr. Ivo Martiník, Ph.D.

Ostrava 2017

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

# Zadání bakalářské práce

Student: **Lukáš Josefus**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R017 Informatika v ekonomice

Téma: Multiplatformní aplikace pro podporu výuky síťové analýzy
Multi-platform Application for Network Analysis Teaching Support

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teoretická východiska práce
3. Analýza současného stavu a požadavků na aplikaci
4. Návrh a implementace aplikace
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

MORAVCOVÁ, Eva a Jitka BAŇAŘOVÁ. *Operační výzkum.* Ostrava: VŠB-TU, Ekonomická fakulta, 2003. ISBN 80-248-0365-8.
ZOUNEK, J., L. JUHAŇÁK, H. STAUDKOVÁ a J. POLÁČEK. *E-learning: učení (se) s digitálními technologiemi.* Praha: Wolters Kluwer, 2016. ISBN 978-80-7552-217-7.
SCHILDT, Herbert. *Java 7: výukový kurz.* Přeložil Lukáš KREJČÍ. Brno: Computer Press, 2012. ISBN 978-80-251-3748-2.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Ivo Martiník, Ph.D.**

Datum zadání: 18.11.2016
Datum odevzdání: 05.05.2017


Ing. Petr Rozehnal, Ph.D.
*vedoucí katedry*

prof. Dr. Ing. Zdeněk Zmeškal
*děkan fakulty*

# Prohlášení

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně.

V Ostravě dne 3. 5. 2017

.....
Lukáš Josefus

# Poděkování

Chtěl bych na tomto místě poděkovat vedoucímu mé bakalářské práce doc. RNDr. Ivo Martiníkovi, Ph.D. za cenné rady a připomínky k této práci a inspiraci pro výběr tématu.

# Obsah

# 1 Úvod

V dnešní době již existuje spousta softwarových nástrojů, které jsou zaměřeny na podporu projektového řízení. Projektový tým tak již nemusí ručně provádět všechny výpočty a sestavovat nejrůznější grafy a diagramy. I přesto je však důležité, aby uživatelé těchto nástrojů znali alespoň základní prvky a metody síťové analýzy a chápali, jak se jimi používané nástroje dostanou k požadovaným výsledkům.

Hlavním cílem této práce je navrhnout a implementovat aplikaci, která by mohla na Ekonomické fakultě VŠB-TUO (v dalším textu jen Ekonomická fakulta) sloužit jako podpůrný nástroj při výuce vybraných oblastí síťové analýzy, konkrétně problematiky síťového grafu a časové analýzy projektu pomocí metody kritické cesty. Aplikace by rovněž měla studentům usnadňovat domácí přípravu na zápočet a zkoušku z této problematiky tím, že jim bude poskytovat zpětnou vazbu a budou tak okamžitě vědět, jestli danou úlohu řeší správně, nebo případně kde udělali chyby. Aplikace by navíc měla být multiplatformní a fungovat především na platformách Windows a macOS. Tento hlavní cíl lze rozdělit do třech následujících dílčích cílů.

Prvním dílčím cílem je provést analýzu současného stavu výuky síťové analýzy na Ekonomické fakultě a analýzu požadavků na aplikaci.

Druhým dílčím cílem je navrhnout a implementovat tuto aplikaci podle stanovených požadavků.

Třetím dílčím cílem je pak praktické ověření aplikace při řešení konkrétních úloh.

Tato práce je rozdělena do 5 kapitol.

Ve druhé kapitole jsou definována její základní teoretická východiska. V první části kapitoly je stručně charakterizován tzv. e-learning, spolu s jeho výhodami a nevýhodami. V dalších částech jsou pak vysvětleny základní pojmy síťové analýzy a stručně popsány technologie a nástroje využité pro vývoj aplikace.

Ve třetí kapitole je provedena analýza současného stavu výuky síťové analýzy na Ekonomické fakultě a také analýza požadavků na aplikaci vytvářenou v rámci této práce.

Obsahem čtvrté kapitoly je samotný návrh a implementace aplikace. Nejprve je zde aplikace popsána z hlediska struktury programu. Následně je popsán návrh jejího uživatelského rozhraní. V dalších částech kapitoly je pak popsána implementace vybraných funkcí. Na konci kapitoly je pak využití vytvořené aplikace demonstrováno na konkrétní úloze.

V poslední kapitole je pak provedeno shrnutí celé práce a porovnání jejích výsledků se stanovenými cíli.

# 2 Teoretická východiska práce

V této kapitole jsou popsána základní teoretická východiska práce a je rozdělena do tří podkapitol. V první podkapitole je stručně charakterizován tzv. e-learning a jeho klady a zápory. Ve druhé podkapitole jsou vysvětleny vybrané základní pojmy síťové analýzy. Ve třetí podkapitole jsou pak popsány vybrané technologie a nástroje, které byly použity ke tvorbě výsledné aplikace.

## 2.1 E-learning

I přesto, že se pojem e-learning používá stále častěji, definice tohoto pojmu se u různých autorů někdy i výrazně liší. Záleží totiž na tom, co všechno daný autor za e-learning považuje a co již nikoliv. V této práci bude vycházeno z definice, která pojem e-learning chápe v jeho širším významu.

Podle Zounek, Juhaňák, Staudková a Poláček (2016, str. 34) e-learning v širším pojetí „*zahrnuje jak teorii a výzkum, tak i jakýkoliv vzdělávací proces s různým stupněm intencionality, v němž jsou používány digitální technologie*“.

Podmnožinou velice širokého pojmu e-learning je pak tzv. blended learning, což je označení pro využívání digitálních technologií (nejen nástrojů, ale také elektronických zdrojů informací apod.) v kombinaci s prezenční výukou, s cílem výuku maximálně zefektivnit. (Zounek, Juhaňák, Staudková a Poláček, 2016)

Zounek, Juhaňák, Staudková a Poláček (2016) ve své knize uvádějí, že podle různých výzkumů považuje většina studentů (dle některých výsledků až 79%) právě blended learning, tedy spojení prezenční výuky s online výukou, za nejlepší způsob výuky.

Stejně jako všechno ostatní, také e-learning má své výhody a nevýhody, které je potřeba při rozhodování o jeho využití důkladně zvážit. Vždy je však potřeba posuzovat tyto výhody a nevýhody přímo v konkrétní situaci, protože závisejí na mnoha podmínkách (finančních, geografických, kulturních apod.). To, co se může za určitých podmínek jevit jako výhoda, může být za jiných podmínek spíše nevýhodou a naopak. (Zounek, Juhaňák, Staudková a Poláček, 2016)

Obecně však za výhodu e-learningu bývá považován například fakt, že díky němu mohou studenti studovat v podstatě kdekoliv a kdykoliv, mají k dispozici informace z mnoha různých zdrojů, nebo že mohou studovat svým vlastním tempem. Určité výhody přináší e-learning také vyučujícím i samotným vzdělávacím institucím. Vyučující mohou například vytvářet studijní materiály v různých formách (textové, audiovizuální aj.), odkazovat studenty na jiné materiály dostupné na internetu, řídit výuku s využitím různých systémů pro řízení výuky apod. (Zounek, Juhaňák, Staudková a Poláček, 2016)

E-learning má však také řadu nevýhod. Jako jedna z jeho hlavních nevýhod bývají uváděny možné vysoké náklady na jeho zavedení, a to jak z pohledu studenta (např. pořízení počítače nebo určitého softwaru), tak z pohledu vzdělávací instituce (např. náklady na pořízení a instalaci hardwaru a softwaru, zaškolení vyučujících, reorganizace způsobu výuky apod.). (Zounek, Juhaňák, Staudková a Poláček, 2016)

Zounek, Juhaňák, Staudková a Poláček (2016) pak také v souvislosti s nevýhodami e-learningu upozorňují na možné zdravotní a psychické problémy způsobené nepřiměřeně dlouhou dobou strávenou u počítačů.

Při zavádění e-learningových prvků (tj. digitálních technologií) do výuky je tedy třeba klást velký důraz především na to, aby tyto prvky byly efektivně využívány a výuku skutečně podporovaly a nebyly pouze jakýmsi jejím zpestřením, nebo ji dokonce narušovaly. (Zounek, Juhaňák, Staudková a Poláček, 2016)

## 2.2 Síťová analýza

„*Síťová analýza je soubor modelů a metod, které vycházejí z grafického vyjádření složitých projektů a provádějí analýzu těchto projektů z hlediska času, nákladů nebo zdrojů nutných k jejich realizaci.*“ (Fiala, 2004, str. 79)

Jinými slovy je síťová analýza nástrojem pro časovou, nákladovou a zdrojovou analýzu projektů.

V této podkapitole budou nejprve uvedeny vybrané základní pojmy teorie grafů, která je teoretickým základem síťové analýzy. Poté bude popsán síťový graf a následně metoda kritické cesty, která slouží k časové analýze projektů.

### 2.2.1 Vybrané základní pojmy teorie grafů

Tato část byla zpracována podle Moravcová a Baňařová (2003) a Vítečková, Přidal a Koudela (2006).

**Graf** je útvar, který je možné pomocí bodů (označovaných jako uzly nebo vrcholy) a spojnic mezi těmito body (tyto spojnice jsou označovány jako hrany), znázornit jako obrázek v rovině (případně v prostoru).

**Konečný graf** je označení pro graf, který je tvořen konečným počtem uzlů a hran.

**Orientovaný graf** je graf, který je tvořen orientovanými hranami. To znamená, že u každé hrany daného grafu je určen také její směr. Uzel, ze kterého orientovaná hrana vystupuje, je označován jako její počáteční uzel. Uzel, do kterého daná orientovaná hrana vstupuje, je pak označován jako její uzel koncový.

**Hranově ohodnocený graf** je graf, jehož každá hrana je ohodnocena číslem.

**Uzlově ohodnocený graf** je graf, jehož každý uzel je ohodnocen číslem.

**Cesta** v grafu je posloupnost hran, které na sebe navazují, přičemž musí platit, že se žádné hrany ani uzly neopakují.

**Cyklus** je cesta v orientovaném grafu, která začíná a končí ve stejném uzlu.

**Acyklický graf** je graf, který neobsahuje cyklus ani smyčky (tj. hrana nesmí vstupovat do uzlu, ze kterého vystupuje). Graf je tedy možné považovat za acyklický právě tehdy, když je možné jeho uzly očíslovat přirozenými čísly tak, aby platilo, že každá hrana vstupuje do uzlu, který má vyšší číslo než uzel, ze kterého daná hrana vystupuje.

**Souvislý graf** je graf, u kterého je možné mezi každou dvojicí uzlů nalézt alespoň jednu cestu.

**Multigraf** je graf, mezi jehož některými dvěma uzly existuje více shodně orientovaných hran.

**Síť** je pak označení pro graf, který je konečný, orientovaný, ohodnocený, acyklický, souvislý, má jeden počáteční a jeden koncový uzel a není multigrafem (multigrafem nesmí být proto, aby byly všechny hrany daného grafu jednoznačně identifikovatelné). Počátečním uzlem sítě se rozumí uzel, do kterého nevstupují žádné hrany (tj. tento uzel má pouze hrany výstupní). Koncovým uzlem sítě se pak naopak rozumí uzel, ze kterého žádné hrany nevystupují. (tj. tento uzel má pouze hrany vstupní). V síťové analýze se pak pro tento typ grafu používá pojem „síťový graf“.

### 2.2.2 Síťový graf

Vlastnosti síťového grafu byly popsány již v předchozí části. V této části tedy bude síťový graf popsán z hlediska svého využití v síťové analýze.

Rosenau (c2007) popisuje síťový graf jako grafické zobrazení, jehož cílem je vyjádřit závislosti mezi projektovými činnostmi a událostmi.

Fiala (2004) pak ve své knize uvádí, že se síťové grafy dělí na hranově a uzlově definované (ohodnocené). V hranově definovaných síťových grafech jsou činnosti představovány orientovanými hranami a uzly znázorňují události asociované se začátky a konci činností. V uzlově definovaných síťových grafech jsou pak činnosti představovány uzly a hrany pouze vyjadřují závislosti mezi těmito činnostmi.

V dalších částech této práce bude vycházeno pouze z hranově definovaného síťového grafu, a proto je zde tento typ popsán podrobněji.

#### Hranově definovaný síťový graf

Jak již bylo zmíněno, u hranově definovaných síťových grafů jsou činnosti znázorněny jako orientované hrany a uzly vyjadřují události, tedy začátky a konce

činností. Jednotlivé hrany (činnosti) jsou pak ohodnoceny údaji o čase (době trvání), nákladech nebo nárocích na zdroje.

Počáteční uzel hrany (uzel, ze kterého daná hrana vystupuje) představuje událost, kdy činnost reprezentovaná danou hranou začíná. Koncový uzel hrany (uzel, do kterého daná hrana vstupuje) pak představuje událost, kdy činnost reprezentovaná danou hranou končí.

Jak již bylo také zmíněno, síťový graf má právě jeden počáteční a jeden koncový uzel. Počáteční uzel síťového grafu (uzel, do kterého nevstupují žádné hrany) představuje začátek daného projektu. Koncový uzel síťového grafu (uzel, ze kterého žádné hrany nevystupují) pak reprezentuje konec daného projektu.

U hranově definovaných síťových grafů je však také potřeba, v některých případech, pro správné zachycení návazností činností využít tzv. fiktivní hrany s nulovým ohodnocením. (Fiala, 2004)

Fiktivní činnost (hrana) má nulovou dobu trvání. Používá se v případech, kdy je potřeba:

- zabránit vzniku multigrafu,
- vyjádřit závislost mezi činnostmi, u kterých nejde tato závislost vyjádřit přímo. (Friebelová, 2006)

V síťovém grafu jsou pak fiktivní hrany většinou znázorněny čárkovaně.

### 2.2.3 Metoda kritické cesty

Metoda kritické cesty neboli CPM (Critical Path Method) je nejrozšířenější metodou síťové analýzy, a to i přesto, že je určena pro analýzu časové náročnosti pouze deterministických projektů (projektů, u kterých přesně známe doby trvání jednotlivých činností), včetně všech jejich dílčích činností. (Moravcová a Baňařová, 2003)

Kritická cesta je označení pro posloupnost činností, jejichž doba trvání přímo ovlivňuje dobu trvání celého projektu. Činnosti, které tvoří kritickou cestu, jsou pak označovány jako kritické činnosti. Kritických cest může být v projektu i několik. (Fiala, 2004; Moravcová a Baňařová, 2003)

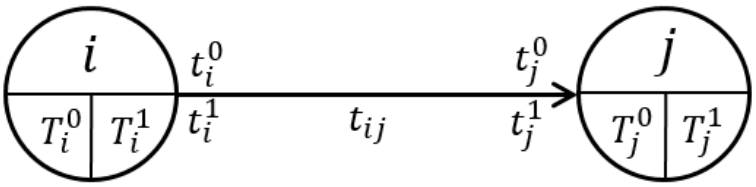
Postup pro analýzu projektu pomocí metody kritické cesty je následující:

- vytvořit hranově definovaný síťový graf projektu,
- vypočítat potřebné časové charakteristiky projektu, včetně jednotlivých činností,
- pro jednotlivé činnosti vypočíst jejich časové rezervy,
- nalézt a analyzovat kritickou cestu. (Moravcová a Baňařová, 2003)

**Výpočty časových charakteristik**

U uzlů se počítají jejich nejdříve možné a nejpozději přípustné aktivační termíny. U hran (činností) se pak počítají termíny jejich nejdříve možného a nejpozději přípustného zahájení a ukončení. Tyto časové charakteristiky jsou znázorněny v obrázku 2.1 a jejich význam bude vysvětlen dále.

*Obrázek 2.1 - Časové charakteristiky, zdroj: vlastní*




Následující pravidla výpočtů časových charakteristik jsou zpracována podle Fiala (2004) a Moravcová a Baňařová (2003). Ve vzorcích jsou použity tyto proměnné („$(i,j)$“ značí činnost mezi uzly $i$ a $j$):

- $t_{ij}$ - doba trvání činnosti $(i,j)$,
- $t_i^0$ - nejdříve možný termín zahájení činnosti $(i,j)$,
- $t_j^0$ - nejdříve možný termín ukončení činnosti $(i,j)$,
- $t_i^1$ - nejpozději přípustný termín zahájení činnosti $(i,j)$,
- $t_j^1$ - nejpozději přípustný termín ukončení činnosti $(i,j)$,
- $T_i^0$ - nejdříve možný aktivační termín uzlu $i$,

- $T_i^1$ - nejpozději přípustný aktivační termín uzlu $i$,
- $T_j^0$ - nejdříve možný aktivační termín uzlu $j$,
- $T_j^1$ - nejpozději přípustný aktivační termín uzlu $j$.

Nejprve se počítají termíny nejdříve možné.

Nejdříve možný aktivační termín každého uzlu se určí jako maximální hodnota z nejdříve možných termínů ukončení činností, které do daného uzlu vstupují, tj. podle vztahu

$$T_j^0 = \max t_j^0. \tag{2.1}$$

Výjimkou je počáteční uzel grafu, jehož nejdříve možný aktivační čas se rovná termínu $T$ (plánovaný termín zahájení projektu, většinou 0), protože představuje začátek projektu, a tedy nemá žádné vstupní hrany (nepředcházejí mu žádné činnosti). Nejdříve možný aktivační termín koncového uzlu grafu pak vyjadřuje nejdříve možný termín dokončení celého projektu.

Nejdříve možné termíny zahájení jednotlivých činností odpovídají nejdříve možným aktivačním termínům jejich počátečních uzlů, tj. určí se podle vztahu

$$t_i^0 = T_i^0. \tag{2.2}$$

Nejdříve možné termíny ukončení jednotlivých činností se pak vypočítají jako součty jejich nejdříve možných termínů zahájení a jejich dob trvání, tj. podle vztahu

$$t_j^0 = t_i^0 + t_{ij}. \tag{2.3}$$

Pokud je po vypočtení všech nejdříve možných termínů nejdříve možný aktivační termín koncového uzlu grafu, tj. nejdříve možný konec projektu, větší než jeho plánovaný termín ukončení, je jeho provedení v plánovaném termínu

nemožné a je potřeba jej znovu naplánovat (případně posunout plánovaný termín ukončení projektu). V opačném případě se pak počítají termíny nejpozději přípustné.

Nejpozději přípustný aktivační termín každého uzlu se určí jako minimální hodnota z nejpozději přípustných termínů zahájení činností, které z daného uzlu vystupují, tj. podle vztahu

$$T_i^1 = \min t_i^1. \tag{2.4}$$

Výjimkou je koncový uzel grafu, jehož nejpozději přípustný aktivační termín se rovná stanovenému termínu $T$ (plánovaný termín ukončení projektu), protože nemá žádné výstupní hrany. U koncového uzlu grafu tedy musí platit, že

$$T_j^1 = T. \tag{2.5}$$

Nejpozději přípustné termíny ukončení jednotlivých činností odpovídají nejpozději přípustným aktivačním termínům jejich koncových uzlů, tj. určí se podle vztahu

$$t_j^1 = T_j^1. \tag{2.6}$$

Nejpozději přípustné termíny zahájení jednotlivých činností se pak vypočítají jako rozdíly mezi jejich nejpozději přípustnými termíny ukončení a jejich dobami trvání, tj. podle vztahu

$$t_i^1 = t_j^1 - t_{ij}. \tag{2.7}$$

**Výpočet časových rezerv**

Nejčastěji se u činností počítají tři typy časových rezerv:

- celkové,
- volné,
- nezávislé.

Výpočty časových rezerv činností jsou opět zpracovány podle Fiala (2004) a Moravcová a Baňařová (2003). Ve vzorcích jsou použity proměnné popsané v předchozí části – „Výpočty časových charakteristik“.

Celkové časové rezervy ($R_c$) činnosti vyjadřují, o kolik časových jednotek je možné prodloužit její dobu trvání nebo posunout termín jejího nejdříve možného zahájení, aniž by byla ovlivněna doba trvání celého projektu. Tuto rezervu je možné spočítat například jako rozdíl mezi jejím nejpozději přípustným a nejdříve možným termínem ukončení, tj. podle vzorce

$$R_c = t_j^1 - t_j^0. \tag{2.8}$$

Volné časové rezervy ($R_v$) činnosti vyjadřují, o kolik časových jednotek je možné prodloužit její dobu trvání nebo posunout termín jejího nejdříve možného zahájení, aniž by byly ovlivněny nejdříve možné termíny zahájení po ní následujících činností. U činnosti může volná časová rezerva vzniknout jen tehdy, když do jejího koncového uzlu vstupuje více činností. Je možné ji spočítat jako rozdíl mezi nejdříve možným aktivačním časem jejího koncového uzlu a jejím nejdříve možným termínem ukončení, tj. podle vztahu

$$R_v = T_j^0 - t_j^0. \tag{2.9}$$

Nezávislé časové rezervy ($R_n$) činnosti pak vyjadřují, o kolik časových jednotek je možné prodloužit její dobu trvání nebo posunout termín jejího nejdříve možného zahájení, aniž by byly ovlivněny nejpozději přípustné termíny ukončení činností, které jí předcházejí, ale také nejdříve možné termíny zahájení činností, které po ní následují. Tuto rezervu je možné vypočítat podle vzorce

$$R_n = T_j^0 - T_i^1 - t_{ij}. \tag{2.10}$$

Protože však nezávislé časové rezervy mohou, na rozdíl od časových rezerv celkových a volných, vyjít záporné, počítají se podle upraveného vzorce

$$R'_n = \max(R_n, 0). \tag{2.11}$$

Mezi těmito třemi typy časových rezerv pak musí platit, že

$$R_c \geq R_v \geq R'_n. \tag{2.12}$$

**Nalezení kritické cesty**

Jak již bylo uvedeno, kritickou cestu tvoří posloupnost tzv. kritických činnosti, jejichž doby trvání přímo ovlivňují dobu trvání celého projektu. Z předchozího textu vyplývá, že kritické činnosti jsou ty činnosti, jejichž celkové časové rezervy jsou nulové. Pro nalezení kritické cesty v projektu tedy stačí vybrat ty činnosti, jejichž celkové rezervy jsou nulové.

Poté, co je kritická cesta nalezena, je možné přistoupit k její analýze. Cílem analýzy kritické cesty je pak z hlediska času posoudit, jak moc kritický analyzovaný projekt je a případně i nalézt způsoby, jak kritičnost daného projektu snížit.

## 2.3 Použité technologie a nástroje

V této podkapitole jsou popsány vybrané nástroje a technologie, které byly využity při návrhu a implementaci aplikace určené jako podpůrný nástroj při výuce vybraných oblastí síťové analýzy.

### 2.3.1 Java

Java je silně typovaný, na platformě nezávislý, objektově orientovaný programovací jazyk. Nezávislost na platformě je umožněna tím, že se zdrojové kódy nekompilují přímo do binárního kódu, ale do tzv. bytekódu, který je poté interpretován virtuálním strojem (Java Virtual Machine). (Rouse, 2016)

Tento programovací jazyk byl vytvořen v roce 1991 týmem ze společnosti Sun Microsystems, původně pod názvem Oak. Název Java získal až po svém přejmenování v roce 1995. Původně se mělo jednat o na platformě nezávislý programovací jazyk, určený ke tvorbě softwaru pro tzv. embedded zařízení (např.

mikrovlnné trouby, dálková ovládání apod.). Postupným vývojem se však Java stala hlavním jazykem pro tvorbu aplikací v prostředí Internetu. (Schildt, 2012)

Java je rozdělena na tři edice:

- Java SE (Java Standard Edition) – pro tvorbu klasických desktopových aplikací,
- Java EE (Java Enterprise Edition) – pro tvorbu aplikací na straně aplikačního serveru,
- Java ME (Java Micro Edition) – pro tvorbu softwaru pro tzv. embedded zařízení. (Rouse, 2016)

Aktuální je verze Java 8, která byla oficiálně vydána v březnu 2014. Vydání verze Java 9 je plánováno na rok 2017. (Rouse, 2016)

### 2.3.2 Objektově orientované programování v jazyce Java

Jak bylo uvedeno v předchozí části, jednou z vlastností programovacího jazyka Java je, že je objektově orientovaný. Objektově orientované programování je založeno třech základních principech, konkrétně na zapouzdření, dědičnosti a polymorfismu. V následujícím textu budou tyto základní principy stručně vysvětleny přímo na programovacím jazyku Java. Nejprve je však uvedeno, co je to třída a objekt.

#### Třída a objekt

Obecně lze třídu charakterizovat jako vzor (předlohu), podle kterého jsou následně vytvářeny objekty. V programovacím jazyku Java má ale pojem třída daleko větší význam, neboť v něm třídy představují datové typy. Ve třídě je možné deklarovat nejen to, jaké atributy (vlastnosti, data) bude mít objekt, který podle ní bude později vytvořen, ale také metody (operace), které budou s těmito daty pracovat. Objekty, které jsou následně podle deklarované třídy vytvářeny (tzv. instanciovány), se nazývají instance třídy. (Schildt, 2012)

#### Zapouzdření

Zapouzdření v objektově orientovaném programování představuje mechanismus, který umožnuje sloučit data a operace, které nad těmito daty

pracují, dohromady a skrýt tato data před okolním světem tak, aby nedošlo k narušení jejich integrity či nesprávnému použití. Tímto sloučením dat a operací pak vznikne jednotka označovaná jako objekt. Základem pro zapouzdření v programovacím jazyce Java je tedy třída, která předepisuje podobu objektu. Data jsou reprezentována jednotlivými atributy, operace jsou pak reprezentovány jednotlivými metodami. Skrýt tyto atributy před okolním světem, nebo jeho částí, je pak možné pomocí tzv. modifikátorů přístupu (viditelnosti). (Schildt, 2012)

**Dědičnost**

„*Dědičnost je proces, kdy jeden objekt získává vlastnosti jiného objektu.*“ (Schildt, 2012, str. 31)

V programovacím jazyce Java je dědičnosti dosahováno na úrovni tříd. Dědičností lze vyjádřit, že je jedna třída (tzv. podtřída) pouze určitou specializovanou verzí jiné třídy (tzv. nadtřídy). Při dědění získává podtřída všechny vlastnosti své nadtřídy. Tím pádem mají i instance této podtřídy všechny vlastnosti, které mají instance její nadtřídy, a také některé vlastnosti navíc. Na základě těchto vztahů dědičnosti (podtřída - nadtřída) pak vznikají hierarchie tříd. Protože je v programovacím jazyce Java, na rozdíl od některých jiných programovacích jazyků, podporována pouze tzv. jednoduchá dědičnost (tj. třída může být podtřídou nejvýše jedné třídy), existuje v ní pouze jediná hierarchie, na jejímž vrcholu je třída `Object` z balíčku `java.lang`. (Schildt, 2012)

**Polymorfismus**

Schildt (2012, str. 30) ve své knize uvádí, že polymorfismus je „*vlastnost, která umožňuje jedno rozhraní pro přístup k obecné třídě akcí.*“

V programovacím jazyce Java je pak polymorfismu dosahováno dvěma způsoby. Prvním z nich je tzv. přetěžování metod. K přetěžování metody dochází v případě, že je v jedné třídě deklarováno více metod se stejným názvem, které se však liší v počtu parametrů nebo jejich datových typech. Druhým případem je pak situace, kdy třída tzv. přepisuje metodu své nadtřídy nebo rozhraní, které implementuje. Přepsání metody znamená, že je v podtřídě vytvořena nová implementace metody deklarované v nadtřídě (nebo rozhraní). Pokud je tedy za

běhu programu provedena určitá metoda nad instancí této podtřídy, může se chovat jinak než v případě, kdy je stejná metoda provedena nad instancí původní třídy. (Schildt, 2012)

### 2.3.3 JavaFX

JavaFX je soubor grafických a mediálních balíčků určených pro tvorbu grafických klientských aplikací. V JavaFX aplikacích je možné tvořit výkonná grafická uživatelská rozhraní, ve kterých je možné využívat také nejrůznější animace, audio, video, nebo i 3D grafiku. (Oracle, 2014)

Protože je již JavaFX součástí Java SE, mohou být aplikace vytvořené v JavaFX spuštěny na všech hlavních desktopových platformách (Windows, Linux a Mac OS X), na kterých lze provozovat běhové prostředí Java Runtime Environment. (Oracle, 2014)

Veškeré grafické elementy, které tvoří uživatelské rozhraní JavaFX aplikace, musejí být součástí struktury nazývané JavaFX Scene Graph (graf scény). JavaFX Scene Graph má na starosti operace spojené s vykreslováním aplikace na obrazovku a optimalizace s tímto vykreslováním spojené. O vše se navíc stará automaticky, což má za následek značnou úsporu kódu potřebného pro vývoj aplikace. (Hommel, 2013)

Jednotlivé elementy JavaFX Scene Graph jsou označovány jako uzly. Na základě jeho stromové struktury jsou pak tyto uzly rozdělovány na větve (mohou obsahovat jiné uzly) a listy (nemohou obsahovat jiné uzly). Uzel, který je na vrcholu této stromové hierarchie, je pak označován jako „root“. (Hommel, 2013)

Základní třídou pro reprezentaci libovolných grafických elementů, se kterými JavaFX Scene Graph pracuje je abstraktní třída `Node` z balíčku `javafx.scene`. Pro reprezentaci uzlů, které v sobě mohou obsahovat jiné uzly, je pak základní třídou abstraktní třída `Parent`, která je podtřídou zmiňované třídy `Node` a nachází se ve stejném balíčku. (Hommel, 2013)

Na všechny uzly je také možné aplikovat nejrůznější efekty a transformace, případně je i stylovat pomocí CSS.

Pro tvorbu grafického uživatelského rozhraní je u JavaFX aplikací možné využít také FXML.

**FXML**

FXML je značkovací jazyk, založený na značkovacím jazyce XML, pomocí něhož je možné deklarovat vzhled uživatelského rozhraní aplikace mimo její zdrojový kód. (Fedortsova, 2014)

I přesto, že je FXML použitelné pro libovolná uživatelská rozhraní, obzvláště se hodí pro tvorbu velkých a komplexních statických struktur, jako jsou například formuláře, ovládací panely nebo tabulky. (Fedortsova, 2014)

FXML je také možné používat i u jiných jazyků využívajících Java Virtual Machine, a navíc je v něm možné psát i skripty, např. pomocí JavaScriptu. (Fedortsova, 2014)

Pro některé vývojáře také může být velkou výhodou, že se deklarace uživatelských rozhraní pomocí FXML nemusí psát ručně, ale lze využít například JavaFX Scene Builder, což je grafický návrhář, který umí FXML kód vygenerovat na základě grafického návrhu ze svého editoru. (Fedortsova, 2014)

### 2.3.4 Java serializace

Serializace objektů je mechanismus, jehož účelem je transformace objektů do sekvencí bytů. Mechanismus, který z těchto sekvencí bytů později znovu vytvoří objekty, se pak nazývá deserializace. (Pitner, ©2001-2003)

Standardní mechanismus pro serializaci a deserializaci objektů v programovacím jazyce Java poskytuje Java Serialization API. (Greanier, 2000)

Greanier (2000) ve svém článku dále uvádí, že tento standardní serializační mechanismus je možné využít třemi způsoby.

V případě výběru prvního způsobu stačí třídu upravit tak, aby byly její instance serializovatelné a zbytek je možné ponechat na výchozím serializačním protokolu. Aby byl objekt serializovatelný, musí třída daného objektu splňovat dvě podmínky. První podmínkou je, že tato třída musí buď přímo implementovat

rozhraní `Serializable` z balíčku `java.io`, nebo musí tuto vlastnost získat pomocí dědičnosti. Toto rozhraní však neobsahuje žádné metody, a slouží tedy pouze pro označení dané třídy jako serializovatelné. Druhou podmínku pak představuje požadavek, že všechny datové položky dané třídy musí být rovněž serializovatelné a ty, které serializovatelné nejsou (nebo nemají být), musí být označeny modifikátorem `transient`. (Greanier, 2000)

V případě zvolení druhého způsobu je opět potřeba třídu upravit tak, aby byly její instance serializovatelné, ale navíc je potřeba drobně upravit výchozí serializační protokol. Úprava serializačního protokolu se v dané třídě provede implementací privátních metod `writeObject`, jejímž parametrem je ukazatel na instanci třídy `ObjectOutputStream` z balíčku `java.io`, a `readObject`, jejímž parametrem je ukazatel na instanci třídy `ObjectInputStream` ze stejného balíčku. Je také nutně zajistit, aby následná deserializace objektu pomocí metody `readObject` probíhala ve stejném pořadí, jako jeho serializace v metodě `writeObject`. Tyto úpravy jsou pak využívány například pro šifrování a dešifrování dat daného objektu, nebo pro přidávání dodatečných dat apod., avšak je možné je využít i k jiným účelům. (Greanier, 2000)

Ve třetím způsobu pak třídy neimplementují zmíněné rozhraní `Serializable`, ale rozhraní `Externalizable` z balíčku `java.io`, které dědí od rozhraní `Serializable`. Toto rozhraní pak již definuje dvě metody, které je nutné v dané třídě implementovat. Při použití tohoto způsobu je pak kompletní proces serializace ponechán na programátorovi. Nevýhodou tohoto způsobu je, že musí vše řešit programátor. Výhodou je však získání absolutní kontroly nad serializací a deserializací. Tento přístup se většinou používá v případech, kdy je výchozí serializační protokol nevyhovující, například při serializaci či deserializaci do různých formátů (např. PDF) apod. (Greanier, 2000)

Nezávisle na zvoleném způsobu se pak samotná serializace provádí pomocí instance třídy `ObjectOutputStream` z balíčku `java.io`, která ji provádí do předem zvoleného výstupního proudu dat (např. do souboru, přenos po síti apod.). Nad touto instancí je pak provedena metoda `writeObject`, která

vykoná serializaci objektu, který je jí předán formou vstupního parametru. (Greanier, 2000)

Pro následnou deserializaci se pak využívá instance třídy `ObjectInputStream` z balíčku `java.io`, která ji provádí z předem zvoleného vstupního proudu dat (např. ze souboru, z přenosu po síti apod.). Nad touto instancí je pak provedena metoda `readObject`, která deserializuje objekt ze vstupního proudu dat. (Greanier, 2000)

### 2.3.5 Unified Modeling Language

Unified Modeling Language, zkráceně UML, je univerzální jazyk určený pro grafickou reprezentaci modelů, většinou objektově orientovaných, systémů. Jazyk jako takový však nezávisí na žádné konkrétní metodice ani životním cyklu. Nejlépe adaptována je ale pro využití tohoto jazyka metodika Unified Process. (Arlow a Neustadt, 2007)

Jazyk UML byl již od svého počátku navrhován tak, aby byl nejen co nejvíce srozumitelný pro jeho uživatele, ale také co nejsnáze implementovatelný v tzv. CASE nástrojích. Bez podpory těchto nástrojů by totiž bylo modelování rozsáhlých softwarových systémů velice obtížné. (Arlow a Neustadt, 2007)

V UML existuje čtrnáct různých druhů diagramů.

Každý z nich znázorňuje model z jiného pohledu. Podle způsobu, jakým pracují s časem, se pak dají tyto diagramy rozdělit na:

- statické – též zvané diagramy struktury, které znázorňují prvky a strukturu systému a jejich vzájemné vztahy,
- dynamické – též zvané diagramy chování, které znázorňují, jak se prvky systému vzájemně ovlivňují a jaký vliv mají na výsledné chování systému. (Arlow a Neustadt, 2007)

V praktické části pak bude uveden jeden ze strukturních diagramů, konkrétně diagram tříd, a proto zde bude u tohoto typu diagramu popsáno, k čemu slouží.

### Diagram tříd a úrovně abstrakce

Jak již bylo zmíněno, diagram tříd je jedním ze strukturních diagramů jazyka UML. Jeho účelem je zachytit, z jakých tříd se bude systém skládat a jaké budou mezi jednotlivými třídami vazby.

Třída v diagramu tříd je znázorněna jako obdélník rozdělený na tři vodorovné pásy. V horním pásu se nachází název třídy. Ve středním pásu se nacházejí atributy dané třídy a ve spodním pak operace, které má tato třída provádět. Ve všech pásech se pak mohou nacházet i tzv. ornamenty, které slouží k podrobnějšímu popisu dané třídy. (Arlow a Neustadt, 2007)

Jedinou položkou diagramu tříd, kterou je potřeba vyplnit vždy, je název třídy. Ostatní položky jsou pak již nepovinné a jejich použití se odvíjí od účelu, ke kterému má výsledný diagram tříd sloužit. (Arlow a Neustadt, 2007)

Při tvorbě diagramu tříd (ale i u jiných diagramů) je tedy nutné rozlišovat, za jakým účelem jsou sestavovány. Většinou se rozlišují podle úrovně abstrakce, s jakou je daný systém modelován.

V podstatě je možné systém modelovat na třech základních úrovních abstrakce. Těmito úrovněmi jsou úroveň analytická (konceptuální), návrhová (designová) a implementační. (Arlow a Neustadt, 2007)

Analytický model představuje model na nejvyšší úrovni abstrakce a vždy by měl být vytvořen v jazyku oboru, pro který je systém tvořen. Jeho účelem je zachytit podstatu navrhovaného systému. Třídy v analytickém modelu, tzv. analytické třídy, znázorňují, jaké třídy by se ve vyvíjeném systému měly objevit. Analytické třídy musí povinně obsahovat název dané třídy (v jazyku daného oboru) a názvy nejdůležitějších atributů, případně i množinu nejdůležitějších operací. (Arlow a Neustadt, 2007)

Návrhový model se tvoří na základě modelu analytického a měl by obsahovat podrobnou specifikaci celého budoucího systému. Některé analytické třídy bývá v návrhovém modelu také potřeba rozdělit na několik menších tříd. V tomto modelu už pak musí mít všechny třídy detailně specifikovány všechny potřebné atributy a metody (v návrhovém modelu se již musí abstraktní operace

z analytické třídy upřesnit do konkrétních metod). Atributy by již tedy měly mít specifikován přesný název, datový typ, viditelnost apod. Metody by měly mít specifikovány všechny parametry a návratové typy. Podle návrhového modelu by pak již mělo být snadné vytvářený systém implementovat. (Arlow a Neustadt, 2007)

Nejkonkrétnější je pak model implementační a většinou je získán až na základě analýzy implementovaného zdrojového kódu. Tento model se tvoří přímo například v případech, kdy chceme na základě tohoto modelu nechat zdrojové kódy vygenerovat. V těchto případech je však potřeba používat některý z CASE nástrojů, který tuto funkci podporuje, a navíc je nutné v tomto modelu explicitně specifikovat také velké množství detailů. (Arlow a Neustadt, 2007)

# 3 Analýza současného stavu a požadavků na aplikaci

Tato kapitola je rozdělena na dvě podkapitoly. V první podkapitole je provedena analýza současného stavu výuky síťové analýzy na Ekonomické fakultě. Ve druhé podkapitole jsou pak specifikovány požadavky na aplikaci vyvíjenou v rámci této práce.

## 3.1 Současný stav

Během studia síťové analýzy na Ekonomické fakultě jsou studenti seznámeni, kromě jiného, také se síťovým grafem a časovou analýzou projektu pomocí metody kritické cesty. Na této fakultě je vyučována pouze hranově definovaná verze síťového grafu. U hranově definovaného síťového grafu je ale většinou třeba využívat také tzv. fiktivní hrany. Fiktivní hrany však mohou být pro začátečníky poněkud matoucí a ti pak nemusí síťový graf sestavit správně.

V současné době nejsou na této fakultě při výuce síťové analýzy využívány žádné softwarové nástroje a vše, tedy i síťový graf, je sestavováno a počítáno ručně na papír. Je tomu tak pravděpodobně proto, že drtivá většina dostupných softwarových nástrojů určených pro projektové řízení využívá pouze uzlově definované síťové grafy. Profesionální nástroje se však pro výuku základních principů ani nehodí, zejména kvůli své komplexnosti.

Zásadní nevýhodou sestavování síťového grafu ručně na papír je tedy fakt, že pokud student udělá chybu, musí, v nejhorším případě, celý graf kreslit znovu. Navíc, aby si mohl student při přípravě na zápočet a zkoušku ověřit, zda tuto problematiku (tj. síťový graf a metodu kritické cesty) správně pochopil, je téměř výhradně odkázán pouze na několik málo příkladů, které se řeší na cvičeních nebo přednáškách. Může si sice vymyslet i své vlastní zadání, ale v tomto případě si nebude moci ověřit správnost svého postupu.

Řešením tohoto problému by mohla být právě aplikace vytvářená v rámci této práce, díky které by studenti nemuseli v případě chyby celý síťový graf (nebo jeho část) pracně překreslovat ručně. Navíc by také měli k dispozici v podstatě neomezené množství zadání k procvičování, a tedy by si prostřednictvím ní i ověřili, jestli pracují správně nebo ne.

## 3.2 Požadavky na aplikaci

Požadavky na aplikaci byly stanoveny na základě analýzy provedené v předchozí kapitole, zkušeností autora se studiem síťové analýzy na této fakultě a připomínek vyučujících.

Jak již bylo zmíněno v úvodu, aplikace by měla studentům (uživatelům) usnadnit studium vybraných oblastí síťové analýzy, konkrétně sestavení síťového grafu podle pravidel vyučovaných na Ekonomické fakultě a časovou analýzu projektu pomocí metody kritické cesty.

Student by tedy měl mít možnost vyplnit si zadávací tabulku, podle které poté sestaví síťový graf a provede v něm výpočty potřebné pro následné nalezení a analýzu kritické cesty. Protože se na Ekonomické fakultě rozdělují činnosti v síťovém grafu do tzv. řádů pomocí tzv. hrano-hranové matice, měl by mít student možnost, ověřit si ještě před sestavením síťového grafu, že rozdělení činností do řádů provedl správně. Student by měl mít také možnost, aby si sestavený graf (včetně všech výpočtů a vyznačení kritické cesty) následně nechal aplikací zkontrolovat. V případě nalezení chyby by se mu aplikace měla snažit poradit, jak tuto chybu opravit. Také by měl být schopen ověřit si, jestli umí správně vypočítat časové rezervy jednotlivých činností, konkrétně časové rezervy celkové, volné a nezávislé.

Výsledná aplikace by rovněž měla podporovat ukládání a načítání vytvořeného projektu do a ze souboru a export projektu do formy obrázků.

Uživatelské rozhraní by mělo být jednoduché, přehledné a snadno ovladatelné. Nemělo by obsahovat rušivé prvky, které by studenta zbytečně rozptylovaly.

Protože i na Ekonomické fakultě jsou studenti, kteří na svých počítačích či noteboocích používají různé operační systémy, měla by být aplikace multiplatformní a podporovat především platformy Windows a macOS.

# 4 Návrh a implementace aplikace

Tato kapitola je rozdělena do čtyř podkapitol a jejím obsahem je popis návrhu a implementace aplikace určené jako podpůrný nástroj při výuce vybraných oblastí síťové analýzy. V první podkapitole je stručně charakterizována struktura aplikace. Ve druhé podkapitole je popsáno uživatelské rozhraní aplikace. Ve třetí podkapitole je vysvětleno, jak aplikace kontroluje správnost síťového grafu a v něm provedených výpočtů pro nalezení a analýzu kritické cesty. Ve čtvrté podkapitole je popsáno ukládání a načítaní vytvořeného projektu do a ze souboru a také jeho export do formy obrázků. V páté kapitole je pak demonstrováno využití aplikace na konkrétní úloze.

## 4.1 Struktura aplikace

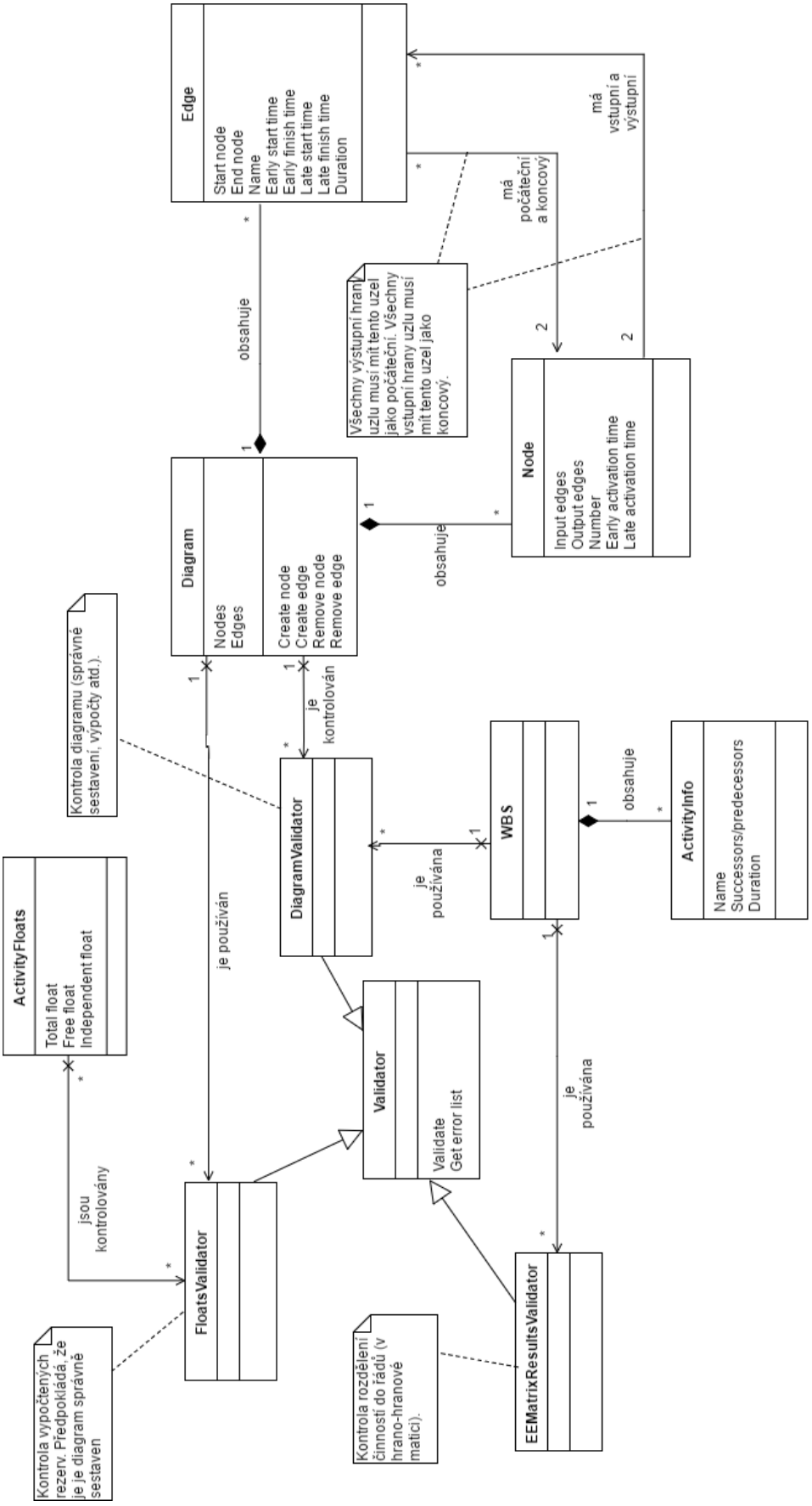
Po provedení analýzy požadavků na aplikaci byl nejprve vytvořen její analytický model. Pro lepší znázornění tohoto modelu byl využit diagram tříd jazyka UML. Diagram tříd představující tento analytický model je zachycen na obrázku 4.1.

Na základě tohoto analytického modelu byl následně vytvořen návrh samotné aplikace. Během návrhu byly některé analytické třídy z tohoto modelu, např. třída `DiagramValidator`, rozděleny do několika dílčích tříd, a samozřejmě bylo potřeba vytvořit i několik dalších pomocných tříd. Několik dalších dodatečných tříd bylo potřeba vytvořit i kvůli rozdělení struktury programu na prezentační a logickou část.

Struktura programu je rozdělena na prezentační a logickou část proto, aby byly výsledné zdrojové kódy přehlednější a snadněji udržovatelné. Prezentační část má na starost vizuální stránku aplikace a komunikaci s uživatelem. Logická část se pak stará o provádění logických operací, např. přidávání či odebírání uzlů a hran v síťovém grafu, jeho kontrolu, apod.

Výchozí grafická reprezentace tříd prezentační části nebyla napsána přímo v programovém kódu, ale byla deklarována s využitím technologie FXML V programovém kódu je pak tato výchozí grafická reprezentace pouze upravována podle aktuálního stavu aplikace.

*Obrázek 4.1 - Analytický model aplikace, zdroj: vlastní*



## 4.2 Návrh uživatelského rozhraní

Jak již bylo popsáno v podkapitole 3.2, aplikace by měla mít jednoduché a přehledné uživatelské rozhraní, bez zbytečných rušivých prvků. Proto byl zvolen systém tzv. záložek. Aby se uživatel nemusel zdržovat zjišťováním, jak jednotlivé záložky otevřít, případně aby nepřišel o rozpracovanou práci kvůli nechtěnému zavření záložky, jsou všechny záložky otevřeny již při vytvoření nového projektu a je u nich zablokována možnost jejich zavření.

Jak je vidět na obrázku 4.2, je uživatelské rozhraní aplikace rozděleno na několik částí.

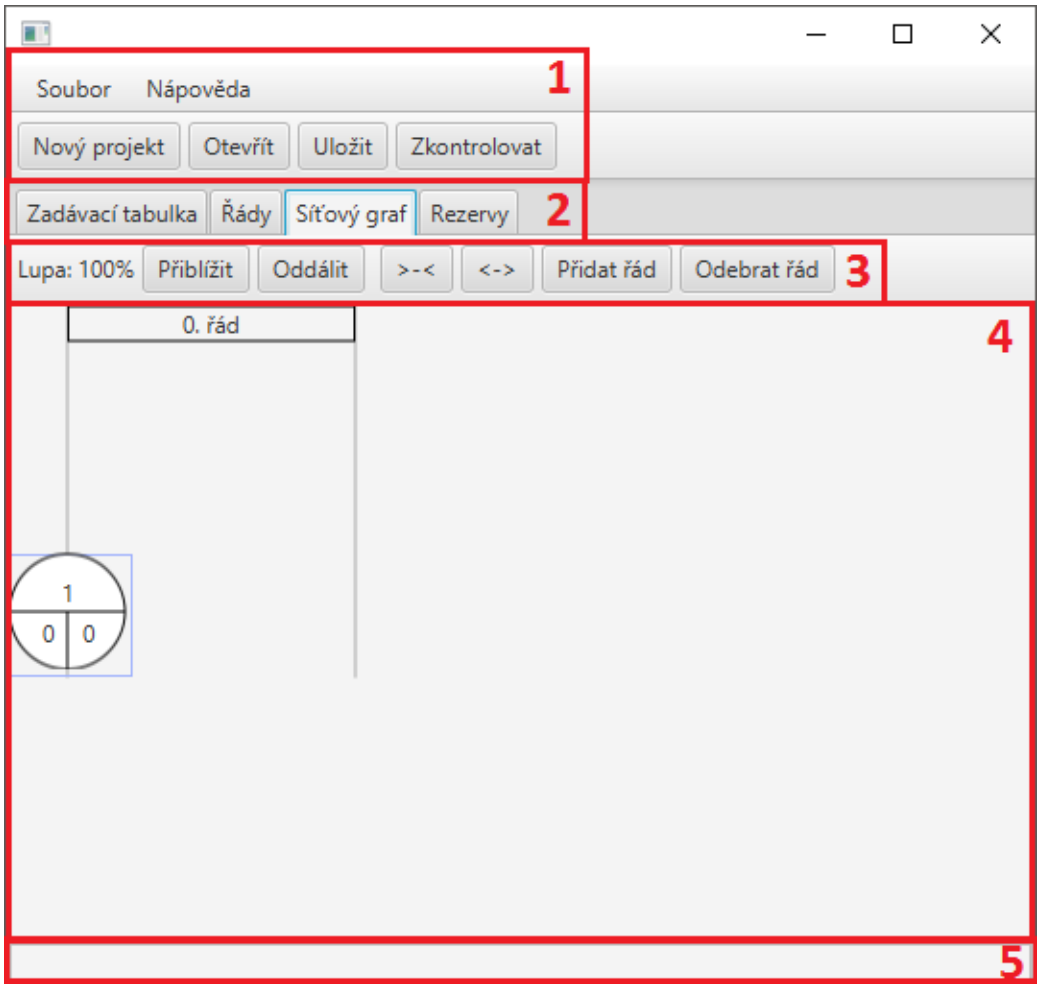
V první části se nacházejí základní ovládací prvky aplikace. Tato část je tudíž společná pro všechny záložky. Poněkud speciální je však tlačítko „Zkontrolovat“. Toto tlačítko je možné použít jen tehdy, když je právě aktivní záložka, jejíž obsah umí tato aplikace zkontrolovat. Jak bude vysvětleno dále, v současné verzi lze toto tlačítko použít na všech záložkách, kromě záložky „Zadávací tabulka“.

Druhá část je tvořena panelem, na kterém lze přepínat mezi jednotlivými záložkami.

Části tři a čtyři se na jednotlivých záložkách liší podle toho, co lze na dané záložce provádět. V tomto případě, tedy na záložce „Síťový graf“, představuje část tři panel nástrojů a část čtyři pracovní plochu, na které je možné sestavit síťový graf a aplikovat metodu kritické cesty. Momentálně aplikace podporuje pouze hranově definovaný síťový graf tak, jak je vyučován na Ekonomické fakultě.

Pátá část pak představuje jednoduchý stavový řádek, na kterém se uživateli zobrazují oznámení o tom, co právě aplikace provádí. Tato část se obzvláště hodí, například při ukládání nebo načítání rozpracovaného projektu do a ze souboru, neboť díky zobrazenému oznámení uživatel pozná, že je aplikace aktivní a ukládá nebo načítá požadovaný soubor. Taková oznámení jsou zobrazována i při vytváření nového projektu, exportování projektu do formy obrázků a při kontrolování obsahu na vybrané záložce.

*Obrázek 4.2 - Uživatelské rozhraní aplikace, zdroj: vlastní*




Aby bylo možné snadno identifikovat, který uzel je aktuálně vybraný a který bude odstraněn při případném stisknutí klávesy *Delete,* je kolem aktuálně vybraného uzlu zobrazen světle modrý rámeček tak, jak je to vidět na obrázku 4.2.

### 4.2.1 Popis jednotlivých záložek

Záložka „Zadávací tabulka“ umožňuje uživateli, jak je již z jejího názvu patrné, vyplnit zadávací tabulku, na jejímž základě pak probíhají jednotlivé kontroly. V tabulce je možné pro každou činnosti určit její název, dobu trvání a činnosti, které by po ní měly následovat. Jak již bylo popsáno, tuto záložku není možné, jako jedinou, zkontrolovat pomocí tlačítka „Zkontrolovat“. Kontrola zadávací tabulky je prováděna již při její tvorbě a to, zda obsahuje cyklus, je ověřeno až při jiných kontrolách.

Na záložce „Řády“ si uživatel může ještě před sestavením síťového grafu ověřit, jestli všechny činnosti správně rozdělil do tzv. řádů.

Záložka „Síťový graf“ uživateli umožňuje síťový graf sestavit a provést v něm všechny výpočty potřebné pro nalezení a analýzu kritické cesty. Kritickou cestu také může v grafu vyznačit. Následně si může pomocí tlačítka „Zkontrolovat“ ověřit, zda síťový graf sestavil správně a jestli jsou správně i výpočty v něm provedené (včetně vyznačení kritické cesty).

Poslední záložkou je záložka „Rezervy“. Uživatel zde vyplní vypočtené časové rezervy jednotlivých činností. Správnost těchto výpočtů si opět může ověřit pomocí tlačítka „Zkontrolovat“.

## 4.3 Kontrola síťového grafu a metody kritické cesty

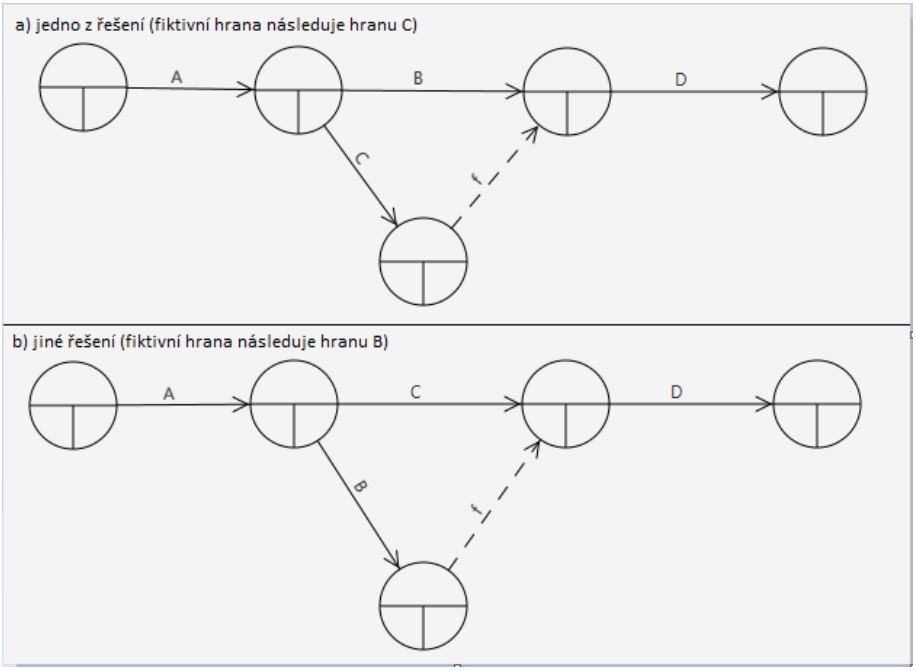
Při této kontrole se ověřuje, jestli je uživatelem sestavený graf v souladu s pravidly pro tvorbu síťového grafu.

Jak bylo popsáno v kapitole 4.2, aplikace umožňuje uživateli sestavení pouze hranově definovaného síťového grafu. Hranově definovaný síťový graf (dále jen síťový graf nebo graf) je však možné, v některých případech, díky fiktivním hranám, sestavit několika způsoby. Například i jednoduchý graf zadaný tabulkou 4.1 lze sestavit více způsoby, jak je vidět na obrázku 4.3.

*Tabulka 4.1 - Ukázka zadávací tabulky, zdroj: vlastní*

| Označení činnosti | Následující činnosti |
|---|---|
| A | B, C |
| B | D |
| C | D |
| D | - |

*Obrázek 4.3- Ukázka více možností sestavení grafu, zdroj: vlastní*




Správnost uživatelem sestaveného síťového grafu tedy nelze kontrolovat pouhým porovnáním s vygenerovaným správným řešením, protože, jak bylo ukázáno výše, správné řešení nemusí být vždy jedinečné. Právě z tohoto důvodu bylo potřeba najít jiný způsob, jak sestavený graf zkontrolovat.

Protože je u síťového grafu potřeba kontrolovat poměrně velké množství údajů (od sestavení až po výpočty), je jeho celková kontrola rozdělena do několika dílčích kontrol. Vzhledem k tomu, že některé z těchto dílčích kontrol vyžadují, aby byla nejprve provedena jiná dílčí kontrola, jsou prováděny v následujícím pořadí:

- kontrola sestavení,
- kontrola rozdělení činností do řádů,
- kontrola očíslování uzlů,
- kontrola výpočtů,
- kontrola kritické cesty.

Pokud některá z těchto dílčích kontrol neproběhne úspěšně, tedy pokud byla nalezena alespoň jedna chyba, jsou uživateli zobrazeny nalezené chyby a další dílčí kontroly již neproběhnou.

Ještě před samotnou celkovou kontrolou je však vždy ověřeno, že mezi činnostmi v zadávací tabulce, podle které se bude graf kontrolovat, neexistuje cyklus. Pokud je cyklus nalezen, je uživateli zobrazena chybová hláška a samotná kontrola již neproběhne.

### 4.3.1 Kontrola sestavení

Tato kontrola je ze všech dílčích kontrol nejdůležitější, protože na ní závisí správná funkčnost několika dalších dílčích kontrol. Cílem této kontroly je totiž podle zadávací tabulky ověřit, jestli na sebe činnosti v uživatelem sestaveném grafu správně navazují, zda jsou správně využity fiktivní hrany apod.

Nejdříve je ověřeno, jestli sestavený graf obsahuje všechny činnosti určené zadávací tabulkou, zda neobsahuje žádné činnosti navíc a jestli se v něm nenachází více hran se stejným názvem. Pokud jsou nalezeny činnosti, které by podle zadávací tabulky v grafu být neměly (a nejsou fiktivní), nebo naopak některé činnosti ze zadávací tabulky v grafu úplně chybí, nebo je nalezeno více hran se stejným názvem, kontrola okamžitě končí a uživateli je zobrazena chybová hláška se seznamem všech chybějících, přebývajících či duplicitních činností. Další kontroly pak již prováděny nejsou, neboť je jasné, že pokud graf obsahuje činnosti navíc nebo v něm některé činnosti chybí, nemůže být sestaven správně.

Následně je provedena kontrola, zda má sestavený graf právě jeden počáteční a jeden koncový uzel. Pokud je v grafu nalezeno počátečních nebo koncových uzlů více, kontrola dále nepokračuje a uživateli je zobrazena chybová hláška. Totéž platí i pro případ, kdy počáteční nebo koncový uzel v grafu naopak zcela chybí.

Pokud se kontrola dostala až do tohoto bodu, je jasné, že sestavený graf obsahuje jen a pouze všechny činnosti stanovené zadávací tabulkou a fiktivní hrany a má právě jeden počáteční a jeden koncový uzel. Až nyní je tedy možné kontrolovat návaznosti jednotlivých činností a správnost použití fiktivních hran.

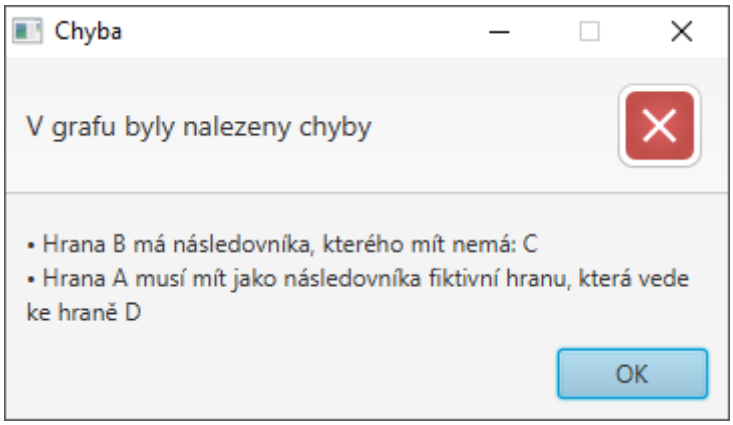
Vzhledem k tomu, že je další průběh této části kontroly poněkud komplikovaný, protože „vše souvisí se vším“, nebude zde podrobněji

rozepisován. Velmi komplikovaná je tato kontrola proto, aby bylo o chybě zjištěno co nejvíce informací a chybové hlášky tak byly pro uživatele přínosné a snažily se mu s chybou co nevíce pomoci.

Pokud například u některé činnosti chybí některá z navazujících činností, uživateli je vypsána přímo tato konkrétní chybějící navazující činnost. Podobná funkčnost je spojena také s činnostmi, které na danou činnost navazují, i když by na ni podle zadávací tabulky navazovat neměly. Navíc, pokud uživatel špatně použije fiktivní hranu, může mu chybová hláška pomoci tuto chybu rychleji odstranit. Na obrázku 4.4 je ukázka takových chybových hlášek.

Pokud jsou nalezeny chyby, je uživateli zobrazena chybová hláška a další kontroly již neproběhnou.

*Obrázek 4.4 - Ukázka chybových hlášek, zdroj: vlastní*




### 4.3.2 Kontrola rozdělení činností do řádů

Cílem této kontroly je ověřit, jestli uživatel správně rozdělil činnosti do jednotlivých řádů.

Aby tato kontrola fungovala správně, musí být nejprve úspěšně provedena kontrola sestavení (viz kapitola 4.3.1).

Nejprve je na základě zadávací tabulky nalezeno správné rozdělení činností do jednotlivých řádů. Pokud se počet řádů v uživatelem sestaveném grafu liší od počtu řádů v nalezeném správném řešení, je uživateli oznámena chyba a dále kontrola nepokračuje.

Pokud jsou počty řádů stejné, jsou postupně procházeny všechny řády sestaveného grafu, a u každého z nich je zjištěno, jestli činnosti, které v něm začínají, odpovídají správnému řešení. V případě nesrovnalostí je uživateli oznámeno, které nalezené činnosti v daném řádu začínat nemají, a naopak, které činnosti by v daném řádu začínat měly.

U každé činnosti, která není fiktivní, je také ověřeno, že se její koncový uzel nachází ve vyšším řádu, než její uzel počáteční.

### 4.3.3 Kontrola očíslování uzlů

Cílem této kontroly je zjistit, jestli uživatel správně očísloval všechny uzly.

Aby tato kontrola fungovala správně, musí být nejprve úspěšně provedena kontrola sestavení (viz kapitola 4.3.1).

V rámci provádění této kontroly je nejprve ověřeno, jestli má počáteční uzel grafu číslo 1 a jestli je číslo koncového uzlu grafu rovnou celkovému počtu uzlů v grafu. Pokud tomu tak není, kontrola končí a uživateli je zobrazena chybová hláška.

Následně je od koncového uzlu grafu rekurzivně postupováno po jednotlivých uzlech směrem k počátečnímu uzlu grafu a každý z nich je zkontrolován. Při kontrole každého uzlu je nejprve ověřeno, jestli už náhodou nebyl nalezen jiný uzel se stejným číslem. Poté je ověřeno, jestli je číslo daného uzlu větší, než čísla všech uzlů v nižším řádu, a zároveň větší, než čísla počátečních uzlů všech hran, které do tohoto uzlu vstupují.

Pokud je během kontroly nalezena chyba, kontrola okamžitě končí a uživateli je zobrazena chybová hláška.

### 4.3.4 Kontrola výpočtů

Cílem této kontroly je zjistit, jestli uživatel správně vypočítal všechny potřebné časové charakteristiky jednotlivých uzlů a hran. U uzlů jsou kontrolovány jejich nejdříve možné a nejpozději přípustné aktivační časy. U hran jsou pak kontrolovány nejdříve možné a nejpozději přípustné časy jejich zahájení a ukončení.

Aby tato kontrola fungovala správně, musí být nejprve úspěšně provedena kontrola sestavení (viz kapitola 4.3.1). Kontrolování výpočtů je pak rozděleno do dvou fází.

V první fázi jsou kontrolovány nejdříve možné aktivační časy uzlů a nejdříve možné časy zahájení a ukončení činností. Nejprve je ověřeno, že je nejdříve možný aktivační čas počátečního uzlu grafu roven nule. Poté je od počátečního uzlu grafu rekurzivně postupováno po jednotlivých uzlech a jejich výstupních hranách směrem ke koncovému uzlu grafu. U každé hrany je zjištěno, jestli se její doba trvání shoduje s hodnotou uvedenou v zadávací tabulce (v případě fiktivních hran je ověřeno, jestli je nulová) a jestli jsou její nejdříve možné časy zahájení a ukončení vypočteny podle pravidel uvedených v kapitole 2.2.3. Postup se vždy zastaví, pokud se narazí na koncový uzel grafu nebo uzel, který ještě nemůže být zkontrolován (uzel, u kterého ještě nebyly zkontrolovány všechny vstupní hrany). Tento uzel bude zkontrolován později, až se k němu příslušný algoritmus kontroly dostane jinou cestou.

Ve druhé fázi jsou naopak kontrolovány nejpozději přípustné aktivační časy uzlů a nejpozději přípustné časy zahájení a ukončení činností. Nejprve je zjištěno, jestli je nejpozději přípustný aktivační čas koncového uzlu grafu roven jeho nejdříve možnému aktivačnímu času. Poté je od koncového uzlu grafu rekurzivně postupováno po jednotlivých uzlech a jejich vstupních hranách směrem k počátečnímu uzlu grafu. V této fázi již u hran není ověřováno, jestli jejich doby trvání odpovídají hodnotám uvedeným v zadávací tabulce, a je ověřováno pouze to, jestli jsou jejich nejpozději přípustné časy zahájení a ukončení vypočteny podle pravidel uvedených v kapitole 2.2.3. Postup se vždy zastaví, pokud se narazí na počáteční uzel grafu nebo uzel, který ještě nemůže být zkontrolován (uzel, u kterého ještě nebyly zkontrolovány všechny výstupní hrany). Tento uzel bude zkontrolován později, až se k němu příslušný kontrolní algoritmus dostane jinou cestou.

Pokud je kdykoliv během této části kontroly nalezena chyba, je kontrola okamžitě ukončena a uživateli je zobrazena chybová hláška. Ukončení kontroly ihned po nalezení první chyby je v tomto případě důležité, protože je velice

pravděpodobné, že tato chyba měla vliv na další výpočty, a uživateli by se tak mohl zobrazit zbytečně dlouhý seznam chyb.

### 4.3.5 Kontrola kritické cesty

Cílem této kontroly je ověřit, jestli uživatel v sestaveném síťovém grafu správně vyznačil kritickou cestu v projektu.

Aby tato kontrola fungovala správně, musí být nejprve úspěšně provedena kontrola výpočtů (viz kapitola 4.3.4).

Samotný algoritmus kontroly kritické cesty je potom již velice jednoduchý (viz zdrojový kód 4.1).

*Zdrojový kód 4.1- Kontrola kritické cesty, zdroj: vlastní*

```
@Override
public boolean validate() {
    resetErrorList();

    boolean isCorrect = true;
    for(Edge edge : diagram.getEdgesUnmodifiable()) {
        if(edge.getEarlyFinish() == edge.getLateFinish()) {
            if(!edge.isCritical()) {
              errors.add("Hrana " + edge.getName() + " nemá žádné

              rezervy, a proto musí být označena jako kritická!");
              isCorrect = false;
            }
        }
        else if(edge.isCritical()) {
            errors.add("Hrana " + edge.getName() + " má nenulové

            celkové rezervy, a proto není kritická!");
            isCorrect = false;
        }
    }

    return isCorrect;
}
```

Jak je ze zdrojového kódu 4.1 patrné, pouze jsou kontrolovány jednotlivé hrany v sestaveném grafu a u každé z nich je ověřeno, jestli její označení (kritická nebo nekritická) odpovídá jejím celkovým časovým rezervám. V případě, že je činnost (hrana) označena jako kritická, ale má nenulové celkové rezervy, je

zaznamenána chyba. Chyba je zaznamenána i v případě, že je činnost (hrana) označena jako nekritická, ale naopak má celkové rezervy nulové.

Pokud je během kontroly nalezena chyba, kontrola nekončí, ale pokračuje dále. Případné chyby jsou uživateli vypsány až všechny najednou, protože se, na rozdíl od výpočtů, navzájem neovlivňují.

## 4.4 Ukládání, načítání a export do formy obrázků

Tato podkapitola je rozdělena na dvě části. V první části je popsáno uložení a načtení projektu do a ze souboru. Ve druhé části je pak popsán export jednotlivých částí projektu do formy obrázků.

### 4.4.1 Uložení a načtení projektu

Každá instance třídy `Edge` (třída reprezentující hranu) obsahuje, kromě jiného, také reference na dvě instance třídy `Node` (třída reprezentující uzel), jednu pro počáteční a jednu pro koncový uzel dané hrany. Každá instance uzlu pak obsahuje reference na všechny instance hran, které do něj vstupují nebo z něj vystupují.

Tento jev, kdy má jedna instance referenci na druhou instanci a ta má referenci zpět na tu první, je označován jako tzv. cyklické reference.

Kvůli cyklickým referencím by však implementace vlastního způsobu ukládání a načítání byla poměrně složitá. Z tohoto důvodu byl využit mechanismus Java serializace, který s cyklickými referencemi umí pracovat. Pro správnou funkčnost tohoto mechanismu však bylo potřeba třídy všech ukládaných objektů upravit tak, aby byly serializovatelné (viz kapitola 2.3.4), a případně v nich vytvořit metody `writeObject` a `readObject`, aby se správně ukládaly a načítaly i hodnoty datových položek s modifikátorem `transient`. Zdrojový kód 4.2 představuje ukázku deklarace těchto metod pro třídu `Node`.

Poté, co byly tyto úpravy provedeny, byla pro snadnější práci s ukládáním a načítáním vytvořena pomocná serializovatelná třída `ProjectWrapper`, která pouze obsahuje všechny jednotlivé části projektu. Nemusí se tak ukládat nebo

načítat každá část projektu zvlášť, ale stačí ukládat nebo načítat pouze instanci této třídy.

*Zdrojový kód 4.2 – Metody* `writeObject` *a* `readObject` *třídy* `Node`*, zdroj: vlastní*

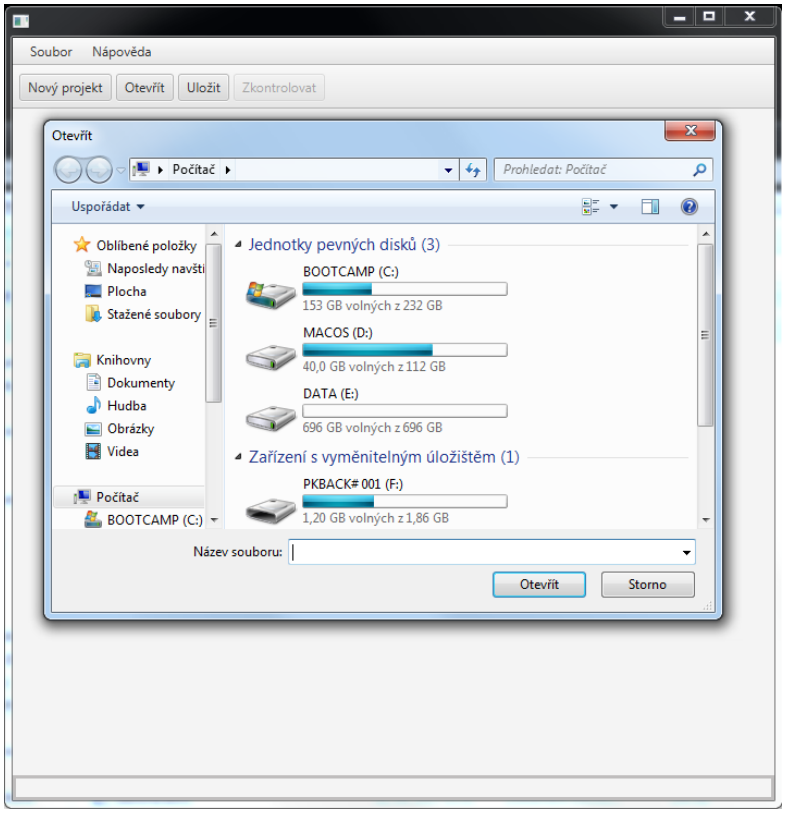
```
private void writeObject(ObjectOutputStream stream) throws
IOException
{
    stream.defaultWriteObject();
    stream.writeDouble(x.get());
    stream.writeDouble(y.get());
    stream.writeInt(nodeNumber.get());
    stream.writeInt(earlyActivation.get());
    stream.writeInt(lateActivation.get());
}

private void readObject(ObjectInputStream stream) throws
IOException, ClassNotFoundException
{
    stream.defaultReadObject();
    x = new SimpleDoubleProperty(stream.readDouble());
    y = new SimpleDoubleProperty(stream.readDouble());
    nodeNumber = new SimpleIntegerProperty(stream.readInt());
    earlyActivation = new SimpleIntegerProperty(stream.readInt());
    lateActivation = new SimpleIntegerProperty(stream.readInt());
}
```

Poté, co byly všechny potřebné třídy připraveny na serializaci a deserializaci, stačilo již jen implementovat funkcionalitu pro samotné ukládání a načítání projektu do a ze zvoleného souboru.

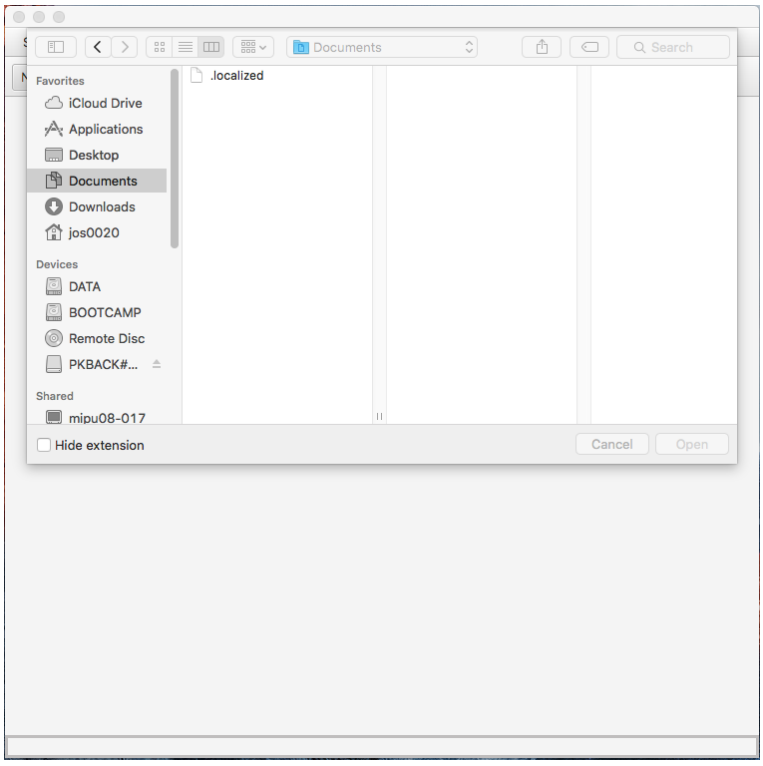
Ještě před samotným ukládáním nebo načítáním je však potřeba získat od uživatele cestu k souboru, do kterého se má projekt uložit nebo ze kterého se má načíst. Pro získání cesty k souboru, byla využita třída `FileChooser` z balíčku `javafx.stage`. Pomocí této třídy je možné uživateli zobrazit klasické dialogové okno pro výběr cesty k souboru, Vzhled tohoto dialogového okna se navíc

přizpůsobuje aktuálnímu operačnímu systému (viz obrázky 4.5 a 4.6), takže uživatel může cestu vybírat tak, jak je zvyklý u jiných programů.

*Obrázek 4.5 – Výběr souboru v MS Windows, zdroj: vlastní*




*Obrázek 4.6 - Výběr souboru v macOS, zdroj: vlastní*



### 4.4.2 Export projektu do formy obrázků

Protože se každá část projektu (zadávací tabulka, rozdělení do řádů, síťový graf, rezervy) ukládá jako samostatný obrázek, je potřeba získat od uživatele cestu k adresáři, do kterého se mají tyto obrázky uložit. Pro získání cesty k adresáři byla využita třída `DirectoryChooser` z balíčku `javafx.stage`, která funguje obdobně jako třída `FileChooser` zmíněná v předchozí části, pouze se místo konkrétního souboru vybírá adresář.

Jak již bylo uvedeno v teoretické části, na vrcholu hierarchie tříd vizuálních elementů, ze kterých je tvořen JavaFX Scene Graph, je třída `Node` z balíčku `javafx.scene`. V této třídě je, kromě jiných, definována i metoda `snapshot`, která vrací grafickou reprezentaci vizuálního elementu, na kterém byla zavolána, ve formě obrázku, konkrétně tedy jako instanci třídy `WritableImage` z balíčku `javafx.scene.image`.

V této aplikaci však třídy, které mají na starost její vizuální stránku, obsahují i oblasti, které ve výsledných obrázcích být nemusí (např. formulář v zadávací tabulce, panel nástrojů v síťovém grafu apod.). Proto byla v těchto třídách vytvořena metoda `getSnapshot`, která pouze zavolá výše zmíněnou metodu `snapshot` na oblasti, která by naopak v obrázku být měla, a vrátí její výsledek. Tím pádem obsahují výsledné obrázky pouze relevantní oblasti (např. tabulku bez formuláře, graf bez panelu nástrojů apod.).

Po získání požadovaných obrázků všech částí projektu je k jejich uložení využita metoda `write` třídy `ImageIO` z balíčku `javax.imageio`. Z formátů (GIF, PNG, JPEG, BMP, WBMP), do kterých umí tato třída ve výchozím nastavení obrázky uložit, byl vybrán formát PNG.

Protože však výše zmíněná metoda `write` neumí pracovat s instancemi třídy `WritableImage`, je potřeba je před ukládáním konvertovat pomocí metody `fromFXImage` třídy `SwingFXUtils` z balíčku `javafx.embed.swing`, která z nich vytvoří instance třídy `BufferedImage` z balíčku `java.awt.image`, se kterými již metoda `write` pracovat umí.

Pro lepší představu popsaného postupu je zde uveden i zdrojový kód (viz zdrojový kód 4.3).

*Zdrojový kód 4.3 – Export do PNG*

```
DirectoryChooser chooser = new DirectoryChooser();
File directory = chooser.showDialog(getScene().getWindow());
if(directory != null) {
  WritableImage wbsImage = wbs.getSnapshot();
  WritableImage diagramImage = diagram.getSnapshot();
  WritableImage floatsImage = floats.getSnapshot();
  WritableImage eeMatrixImage = eeMatrix.getSnapshot();

  File wbsImageFile =
         new File(directory, "tabulka" + System.nanoTime() + ".png");
  File diagramImageFile =
         new File(directory, "graf" + System.nanoTime() + ".png");
  File floatsImageFile =
         new File(directory, "rezervy" + System.nanoTime() + ".png");
  File eeMatrixImageFile =
         new File(directory, "rady" + System.nanoTime() + ".png");

  try {
    ImageIO.write(
      SwingFXUtils.fromFXImage(wbsImage, null), "png", wbsImageFile
    );
    ImageIO.write(
      SwingFXUtils.fromFXImage(diagramImage, null),"png",diagramImageFile
    );
    ImageIO.write(
      SwingFXUtils.fromFXImage(floatsImage, null), "png", floatsImageFile
    );
    ImageIO.write(
      SwingFXUtils.fromFXImage(eeMatrixImage, null), "png",
         eeMatrixImageFile
    );

    showGeneralDialog("Obrázky uloženy", directory.toString());
  }
  catch (IOException ex) {
    System.err.println(ex);
    showErrorDialog("", "Některé obrázky se nepodařilo exportovat");
  }
```

## 4.5 Demonstrace využití aplikace na vybrané úloze

V této podkapitole bude využití vytvořené aplikace (v dalším textu jen aplikace) prakticky demonstrováno na konkrétní úloze. Jako zadání této konkrétní úlohy bude sloužit zadávací tabulka zachycená na obrázku 4.7.

*Obrázek 4.7 - Zadávací tabulka, zdroj: vlastní*

| Činnost | Následovníci | Doba trvání |
|---|---|---|
| A | B,C | 2 |
| B | D,E,F | 3 |
| C | D,E,F,G | 4 |
| D | I | 3 |
| E | I | 1 |
| F | H | 6 |
| G | J | 2 |
| H | I | 8 |
| I | | 1 |
| J | K | 6 |
| K | | 3 |
| | | |

Jednu z funkcí aplikace, konkrétně export do formy obrázků (viz kapitola 4.4.2), demonstruje již samotný obrázek, který byl pomocí aplikace vytvořen (nebyly v něm prováděny žádné dodatečné úpravy, pouze byly oříznuty prázdné řádky tabulky) a zachycuje tedy zadávací tabulku tak, jak ji uživatel v této aplikaci vyplnil. Samotná tabulka se pak skládá ze tří sloupců a vyplňuje se prostřednictvím jednoduchého formuláře, který se nachází vpravo od ní. Zadávací tabulku s formulářem je možné vyplnit na záložce „Zadávací tabulka“.

Jak bylo v této práci již několikrát zmíněno, právě na základě takto sestavené zadávací tabulky jsou následně prováděny všechny potřebné kontroly (rozdělení činností do řádů; sestavení a výpočty v síťovém grafu; výpočet rezerv). Sestavit (nebo upravit) zadávací tabulku je možné kdykoliv, a tedy nemusí být nutně sestavována jako první. Je tak pouze na uživateli, jestli si nejprve vyplní zadávací tabulku, nebo například sestaví síťový graf. Pro správné fungování kontrol je důležité pouze to, aby byla zadávací tabulka vyplněna před jejich spuštěním. To, jestli existuje mezi činnostmi v zadávací tabulce cyklus, je ověřeno před každou kontrolou. V případě nalezení cyklu se pak uživateli zobrazí chybová hláška, která jej vyzve k opravě zadávací tabulky.

Jedním z požadavků na aplikaci bylo, aby si uživatel (student) mohl ještě před vytvořením síťového grafu ověřit, zda činnosti na základě tzv. hrano-hranové matice správně rozdělil do tzv. řádů. Bylo dohodnuto, že bude dostatečné, když si bude moci ověřit až samotné výsledky. K tomuto účelu byla

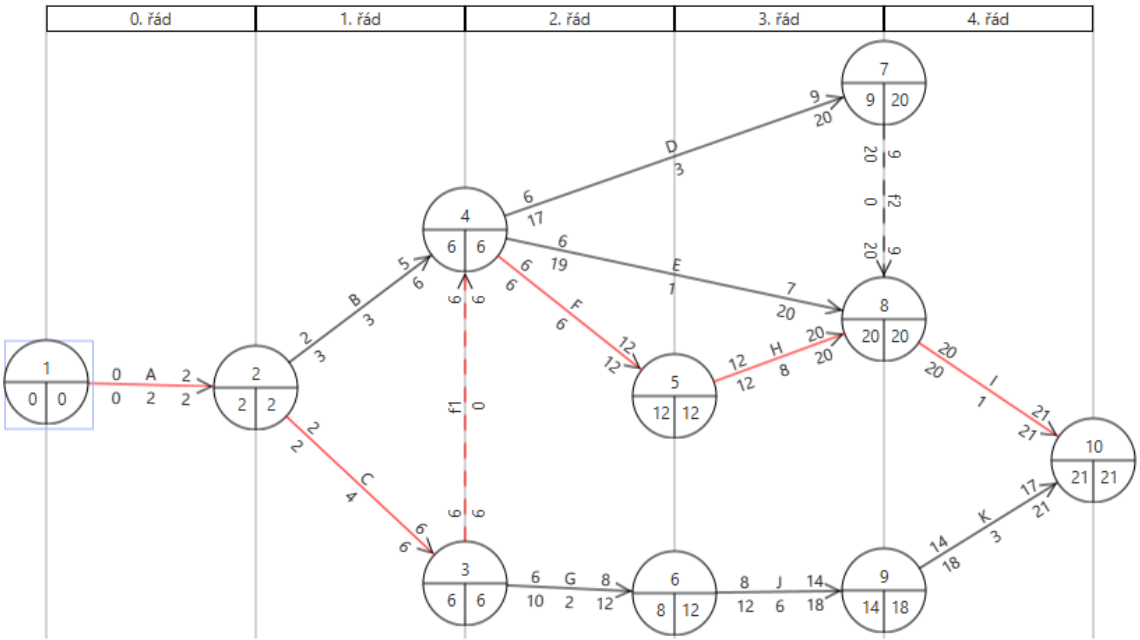
vytvořena jiná tabulka, která se spolu s formulářem pro její vyplnění nachází na záložce „Řády“. Tuto tabulku si již student může nechat aplikací zkontrolovat. Aplikace mu pak v případě nalezení chyb oznámí, které činnosti se nacházejí ve špatném řádu, jestli nějaké řády nebo činnosti chybí apod. V případě, že činnosti do řádů rozdělil správně, v tomto případě tedy tak, jak je to uvedeno na obrázku 4.8 (tento obrázek byl také vytvořen přímo aplikací, byly pouze opět oříznuty prázdné řádky tabulky), aplikace mu pouze zobrazí oznámení o úspěšné kontrole.

*Obrázek 4.8 - Tabulka s řády, zdroj: vlastní*

| Řád | Činnosti |
|---|---|
| 0 | A |
| 1 | B,C |
| 2 | D,E,F,G |
| 3 | H,J |
| 4 | I,K |

Dále si může uživatel v aplikaci jednoduchým způsobem sestavit síťový graf a aplikovat na něm metodu kritické cesty. Síťový graf, včetně aplikace zmíněné metody, si pak může uživatel opět nechat aplikací zkontrolovat. Ukázku toho, jak by v aplikaci vypadal sestavený síťový graf pro tuto konkrétní úlohu zachycuje obrázek 4.9. Tento obrázek byl opět vytvořen prostřednictvím samotné aplikace.

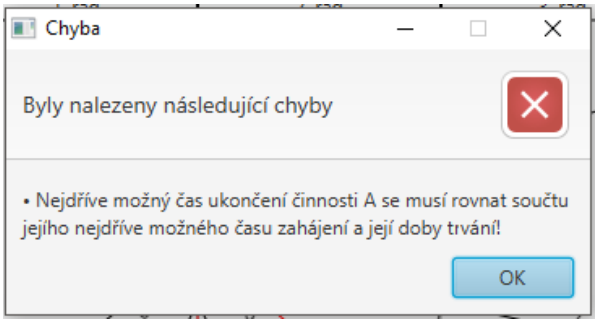
*Obrázek 4.9 - Síťový graf, zdroj: vlastní*




Jak je ze zmíněného obrázku (4.9) patrné, uživatel může v této aplikaci sestavit velice přehledný síťový graf, ve kterém může provést všechny potřebné

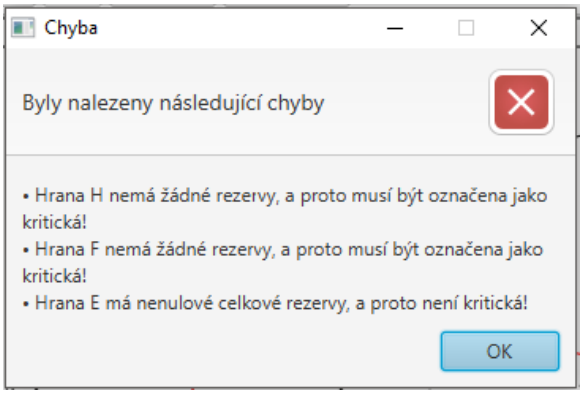
výpočty metody kritické cesty. Kritickou cestu pak také může v sestaveném síťovém grafu označit červenou barvou. Čárkovanými čarami jsou pak znázorněny již zmiňované fiktivní činnosti.

*Obrázek 4.10 - Chybný výpočet, zdroj: vlastní*




V případě, že si uživatel nechá sestavený síťový graf aplikací zkontrolovat a nejsou nalezeny žádné chyby, zobrazí se mu pouze oznámení o úspěšné kontrole. Naopak v případě, kdy je nalezena chyba (nebo chyby), zobrazí se mu chybová hláška, která by se jej měla snažit navést k tomu, jak nalezenou chybu (nebo chyby) opravit. Kdyby uživatel v této konkrétní úloze například zadal špatný nejdříve možný čas ukončení činnost „A“ (správně je 2), aplikace by mu zobrazila chybovou hlášku zachycenou na obrázku 4.10. V případě chyb ve výpočtech se mu vždy zobrazí pouze první nalezená chyba, aby nebyl seznam chyb příliš dlouhý (je totiž velmi pravděpodobné, že tato chyba měla vliv na další výpočty). Jiná je pak situace v případě, kdy by uživatel třeba špatně vyznačil kritickou cestu, v této úloze například místo činností „F“ a „H“ pouze činnost „E“, byly by mu v chybové hlášce zobrazeny všechny nalezené chyby (viz obrázek 4.11). Obdobné chybové hlášky se pak uživateli zobrazují také v případech, kdy jsou nalezeny chyby v rozdělení činností do řádů, očíslování uzlů nebo návaznostech činností (jedna taková chybová hláška, avšak pro jinou úlohu, byla zachycena např. na obrázku 4.4).

*Obrázek 4.11 - Chybná kritická cesta, zdroj: vlastní*

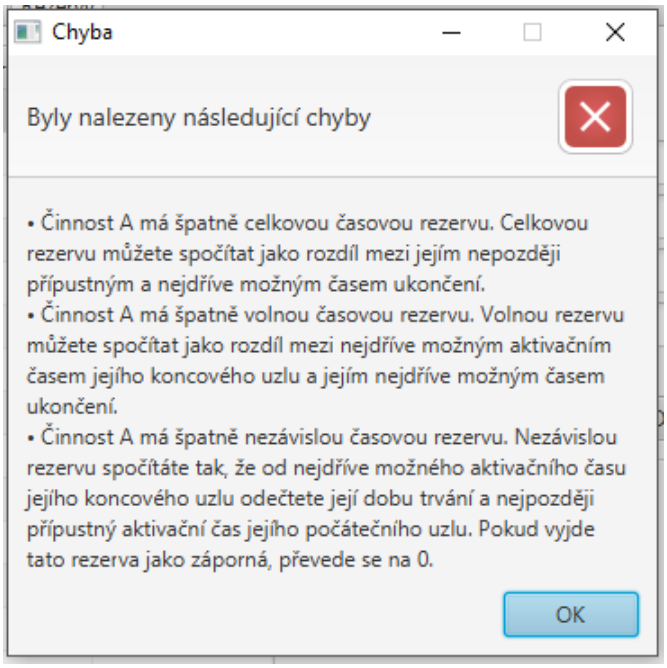


Nakonec si může uživatel také ověřit výpočty časových rezerv jednotlivých činností (dále jen rezervy). Tyto rezervy je možné vyplnit do tabulky, která se spolu s formulářem pro její vyplnění nachází na záložce „Rezervy“. Tabulka s vyplněnými rezervami pro tuto konkrétní úlohu je zachycena na obrázku 4.12 a tento obrázek byl také vytvořen prostřednictvím aplikace (opět pouze s oříznutím prázdných řádků tabulky). Aby bylo možné provést kontrolu těchto rezerv, musí uživatel nejprve vyplnit zadávací tabulku a správně sestavit síťový graf. Pokud je vyžádána kontrola rezerv, ale síťový graf není sestaven správně, je uživatel vyzván k jeho opravě. V případě, že by byla během následné kontroly rezerv nalezena chyba, zobrazila by se uživateli chybová hláška s návodem, jak se konkrétní typ rezerv počítá. Pokud by například uživatel v této úloze špatně vypočítal všechny časové rezervy činnosti „A“, zobrazila by se mu chybová hláška zobrazená na obrázku 4.13.

*Obrázek 4.12 - Časové rezervy, zdroj: vlastní*

| Činnost | Celková rezerva | Volná rezerva | Nezávislá reze... |
|---|---|---|---|
| A | 0 | 0 | 0 |
| B | 1 | 1 | 1 |
| C | 0 | 0 | 0 |
| D | 11 | 0 | 0 |
| E | 13 | 13 | 13 |
| F | 0 | 0 | 0 |
| G | 4 | 0 | 0 |
| H | 0 | 0 | 0 |
| I | 0 | 0 | 0 |
| J | 4 | 0 | 0 |
| K | 4 | 4 | 0 |
| f1 | 0 | 0 | 0 |
| f2 | 11 | 11 | 0 |

*Obrázek 4.13 - Chybné časové rezervy, zdroj: vlastní*




V aplikaci je také možné uložit si takto vytvořený projekt do souboru a následně si jej v aplikaci opět otevřít.

# 5 Závěr

Hlavním cílem této práce bylo navrhnout a implementovat multiplatformní aplikaci, která by mohla na Ekonomické fakultě VŠB-TUO (dále jen Ekonomická fakulta) sloužit jako podpůrný nástroj při výuce vybraných oblastí síťové analýzy, a také při domácí přípravě studentů na zápočet a zkoušku z těchto oblasti. Jako konkrétní oblasti síťové analýzy, které by tato aplikace měla podporovat, byly vybrány problematika síťového grafu a metoda kritické cesty. Požadavkem bylo, aby aplikace fungovala na platformách Windows a macOS (dříve Mac OS X). Pro splnění hlavního cíle pak bylo potřeba splnit tři cíle dílčí.

Nejprve byla provedena analýza současného stavu výuky síťové analýzy na Ekonomické fakultě, během které bylo zjištěno několik nedostatků. Prvním z těchto nedostatků je nutnost tvorby síťových grafů ručně na papír. Pokud student během sestavování síťového grafu udělá chybu, musí jej celý (nebo jeho část) pracně překreslovat. Druhým nedostatkem je pak skutečnost, že je student při přípravě na zápočet a zkoušku odkázán téměř výhradně na příklady z přednášek a cvičení. Na základě výsledků této analýzy pak byly stanoveny a analyzovány požadavky na tvořenou aplikaci. První dílčí cíl, tj. analyzovat současný stav výuky síťové analýzy na Ekonomické fakultě a požadavky na aplikaci, byl tedy splněn.

Následně byl na základě stanovených požadavků vytvořen analytický model aplikace, pro jehož lepší zachycení byl využit diagram tříd jazyka UML. Na základě analytického modelu pak byla aplikace podrobněji navržena z hlediska její struktury, funkčnosti a uživatelského rozhraní. V této fázi byla aplikace také průběžně implementována a testována. Pro implementaci aplikace byl využit programovací jazyk Java a jeho knihovna JavaFX, určená pro tvorbu multiplatformních aplikací s grafickým uživatelským rozhraním. Tyto technologie byly vybrány, protože, jak bylo uvedeno v teoretické části této práce, aplikace vytvořené v JavaFX by měly fungovat na všech hlavních desktopových platformách (tj. Windows, Max OS X a Linux), na kterých lze provozovat běhové prostředí Java Runtime Environment. Po implementaci byla aplikace úspěšně otestována na obou požadovaných platformách a dá se předpokládat, že bude navíc fungovat i na některých distribucích operačního systému Linux.

Ve vytvořené aplikaci je studentovi umožněno, aby si v ní sestavil síťový graf (tak jak je vyučován na Ekonomické fakultě), aplikoval na něm metodu kritické cesty a vypočítal vybrané časové rezervy činností. Vše si pak může nechat aplikací zkontrolovat podle zadávací tabulky, kterou si může v aplikaci rovněž vyplnit, a zjistit tak, zda danou úlohu vyřešil správně, případně kde udělal chyby. Aplikace rovněž podporuje ukládání a načítání vytvořeného projektu do a ze souboru a také jeho export do formy obrázků. Druhý dílčí cíl, tj. navrhnout a implementovat aplikaci podle stanovených požadavků, byl tedy splněn také.

Posledním dílčím cílem pak bylo otestovat funkčnost aplikace při řešení konkrétních úloh. Funkčnost aplikace byla na vybrané úloze demonstrována v praktické části této práce. Kromě průběžného testování funkčnosti na různých úlohách během vývoje byla nakonec funkčnost aplikace také úspěšně ověřena i jedním z vyučujících předmětu Operační výzkum. Třetí dílčí cíl byl tedy rovněž splněn.

Protože byly splněny všechny tři stanovené dílčí cíle, je možné považovat za splněný také hlavní cíl této práce. Vytvořená aplikace splňuje všechny stanovené požadavky a měla by být plně funkční. Nyní je aplikace připravena na otestování přímo ve výuce.

# Seznam použité literatury

## Knižní zdroje

ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

FIALA, Petr. Projektové řízení: modely, metody, analýzy. Praha: Professional Publishing, 2004. ISBN 80-86419-24-x.

MORAVCOVÁ, Eva a Jitka BAŇAŘOVÁ. Operační výzkum. Ostrava: VŠB - TUO, Ekonomická fakulta, 2003. ISBN 80-248-0365-8.

ROSENAU, Milton D. Řízení projektů. Vyd. 3. Brno: Computer Press, c2007. Business books. ISBN 978-80-251-1506-0

SCHILDT, Herbert. Java 7: výukový kurz. Přeložil Lukáš KREJČÍ. Brno: Computer Press, 2012. ISBN 978-80-251-3748-2.

ZOUNEK, Jiří, Libor JUHAŇÁK, Hana STAUDKOVÁ a Jiří POLÁČEK. E-learning: učení (se) s digitálními technologiemi. Praha: Wolters Kluwer, 2016. ISBN 978-80-7552-217-7.

## Elektronické zdroje

FEDORTSOVA, Irina. Mastering FXML: Why Use FXML. In: Oracle [online]. 2014 [cit. 2017-04-16]. Dostupné z: http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm

FRIEBELOVÁ, Jana. *Síťová analýza* [online]. 2006 [cit. 2017-04-15]. Dostupné z: http://www2.ef.jcu.cz/~jfrieb/rmp/data/teorie_oa/SITOVA%20ANALYZA.pdf

GREANIER, Todd. Flatten your objects: Discover the secrets of the Java Serialization API. In: *JavaWorld* [online]. 2000 [cit. 2017-04-16]. Dostupné z: http://www.javaworld.com/article/2076120/java-se/flatten-your-objects.html

HOMMEL, Scott. Working with the JavaFX Scene Graph. In: Oracle [online]. 2013 [cit. 2017-04-16]. Dostupné z: http://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm

ORACLE. JavaFX: Getting Started with JavaFX. [online]. 2014 [cit. 2017-04-17]. Dostupné z: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm

PITNER, Tomáš. Programování v jazyce Java: Serializace objektů [online]. ©2001-2003 [cit. 2017-04-16]. Dostupné z: http://www.fi.muni.cz/~tomp/slides/pb162/foil257.html

ROUSE, Margaret. Java. In: TechTarget [online]. 2016 [cit. 2017-04-17]. Dostupné z: http://searchmicroservices.techtarget.com/definition/Java

VÍTEČKOVÁ, Miluše, Petr PŘIDAL a Tomáš KOUDELA. Základy teorie pravděpodobnosti a teorie grafů [online]. Ostrava, 2006 [cit. 2017-04-24]. Dostupné z: http://books.fs.vsb.cz/SystAnal/texty/download/cele.pdf

# Seznam zkratek

| | |
|---|---|
| 3D | trojrozměrný |
| API | Application Programming Interface |
| BMP | Windows Bitmap |
| CASE | Computer Aided Software Engineering |
| CPM | Critical Path Method |
| CSS | Cascading Style Sheets |
| GIF | Graphics Interchange Format |
| Java SE | Java Standard Edition |
| Java EE | Java Enterprise Edition |
| Java ME | Java Micro Edition |
| JPEG | Joint Photographic Experts Group |
| OS | Operating System |
| PDF | Portable Document Format |
| PNG | Portable Network Graphics |
| UML | Unified Modeling Language |
| VŠB-TUO | Vysoká škola báňská – Technická univerzita Ostrava |
| WBMP | Wireless Bitmap |
| XML | Extensible Markup Language |

# Prohlášení o využití výsledků bakalářské práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;

- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, bakalářskou práci užít (§ 35 odst. 3);

- souhlasím s tím, že bakalářská práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího bakalářské práce. Souhlasím s tím, že bibliografické údaje o bakalářské práci budou zveřejněny v informačním systému VŠB-TUO;

- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;

- bylo sjednáno, že užít své dílo, bakalářskou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 3. 5. 2017

.....Josefus.....

Lukáš Josefus

# Seznam příloh

Příloha č. 1: DVD s aplikací a jejím zdrojovým kódem

## Příloha č. 1: DVD s aplikací a jejím zdrojovým kódem

Na přiloženém DVD se nachází:

- aplikace (soubor Aplikace.jar),
- návod k použití aplikace (soubor navod.pdf),
- zdrojové kódy aplikace ve formě projektu integrovaného vývojového prostředí NetBeans 8.0.2 (složka Aplikace).